Workload Characterization of CFD Applications Using Partial Differential Equation Solvers

Abdul Waheed and Jerry Yan[†]

NAS Technical Report NAS-98-011 March '98

{waheed,yan}@nas.nasa.gov
NAS Parallel Tools Group
NASA Ames Research Center
Mail Stop T27A-2
Moffett Field, CA 94035-1000

Abstract

Workload characterization is used for modeling and evaluating computing systems at different levels of detail. We present workload characterization for a class of Computational Fluid Dynamics (CFD) applications that solve Partial Differential Equations (PDEs). This workload characterization focuses on three high performance computing platforms: SGI Origin2000, IBM SP-2, and a cluster of Intel Pentium Pro based PCs. We execute extensive measurement-based experiments on these platforms to gather statistics of system resource usage, which lead to a quantitative workload characterization. Our workload characterization approach yields a coarse-grain resource utilization behavior that is being applied for performance modeling and evaluation of distributed high performance metacomputing systems. In addition, this study enhances our understanding of interactions between PDE solver workloads and high performance computing platforms and is useful for tuning applications belonging to this class.

[†] MRJ Technology Solutions, Inc., NASA Contract NAS2-14303, Moffett Field, CA 94035-1000

1 Introduction

Many high performance computing (HPC) systems are being developed with commodity microprocessors and network fabrics for improved price/performance ratios. This trend suggests that many HPC architectures are converging with only differences in how memory is accessed [14]. While hardware technologies advance at a rapid pace, software technologies for such systems are lagging behind and becoming major performance bottlenecks [17]. In order to design software environments for future HPC systems, it is essential to understand the interactions between system software and hardware resources.

Workload characterization can be considered as an effort to generalize the results of various measurement-based studies of a class of applications on a specific system. System architects, software developers, and performance analysts can use workload characterization to glean insight about new architectural features, system management protocols, and their impact on performance, respectively. Our objective is to characterize the workload for Computational Fluid Dynamics (CFD) applications that solve partial differential equations. Workload characterization can be particularly beneficial for modeling and detailed evaluation of alternative scheduling and resource management system designs for metacomputing environments [4,11]. Workload characterization is also used by computer architects to study the performance of a proposed system design under realistic operating conditions. Workloads used for such studies are often based on executing actual programs [21] or traces of actual programs [9]. However, this type of fine-grained workload studies conducted in a bottom-up manner (i.e., starting from low-level information) are not only an overkill but also impractical for system users, programmers, and analysts to plan and design software components that use or manage a computing system. Traditional measurement-based workload studies depend on collecting large volumes of accounting log data, which are simplied and analyzed using clustering techniques [7,15]. Therefore, the researchers are considering hierarchical workload characterization methodologies to suit the required level of detail for a particular application [12].

Our workload characterization addresses the issue of level of detail by relying on *a priori* knowledge of system architecture and workload behavior. In particular:

1. we characterize a workload in terms of its requirements for available system resources that may be shared among multiple workloads and managed according to well-defined policies and protocols; and
2. we keep the characterization process focused through a top-down approach, which is based on domain-specific knowledge about a workload (i.e., a class of applications).

These system resources typically include CPU, memory hierarchy, interconnecting network (bus, ethernet, custom network, etc.), and I/O devices. These resources may be shared among multiple workloads and managed according to well-defined policies and protocols. A workload uses the system resources by occupying and releasing them according its characterization and resource management policies imposed by the system. Conventional bottom-up workload studies rely on detailed accounting information about system resource usage over an extended period of time. Such studies are accomplished on the cost of instrumenting the system software and applications, collecting and maintaining large volumes of data, performance overhead to the applications, and enormous amount of time and effort to analyze these data. Using a top-down characterization approach, we need to measure only a subset of the entire resource usage accounting information to capture the level of detail that is necessary for a specific performance study.

We focus on three HPC platforms: SGI Origin2000 with MIPS R10000 nodes; IBM SP-2 with RS/6000 nodes; and a cluster of PCs with Intel Pentium Pro nodes. We use the NAS parallel benchmark (NPB) suite as representative CFD solvers for this characterization effort [2]. Although these benchmarks are not complete CFD applications, they represent generic implementations of partial differential (Navier-Stokes) equation (PDE) solvers that can be found in most real CFD applications. Since these solvers account for the major computational portion of workload, it is appropriate to use them to characterize the interactions between these solvers and two main system resources that they stress: CPU and memory subsystem.

Section 2 of this paper presents an overview of CFD applications based on PDE solution and their distinguishing qualitative characteristics. We present an outline of our workload characterization scheme in Section 3. Section 4 details measurement-based workload characterization on three platforms of interest. Finally, Section 5 presents two scenarios where we are applying this workload characterization. We conclude in Section 6 with a discussion of contributions and future directions of this research.

2 Background

A CFD application is a special case of a scientific application, distinguished due to its large memory requirements. It is a well-known fact that CFD applications stress the system memory in general and caches in particular. Figure 1 presents an overview of numerical solution for typical CFD problems. Numerical solution of partial differential equations that represent the dynamics of a physical system results in the computational workload that we intend to characterize in this paper.

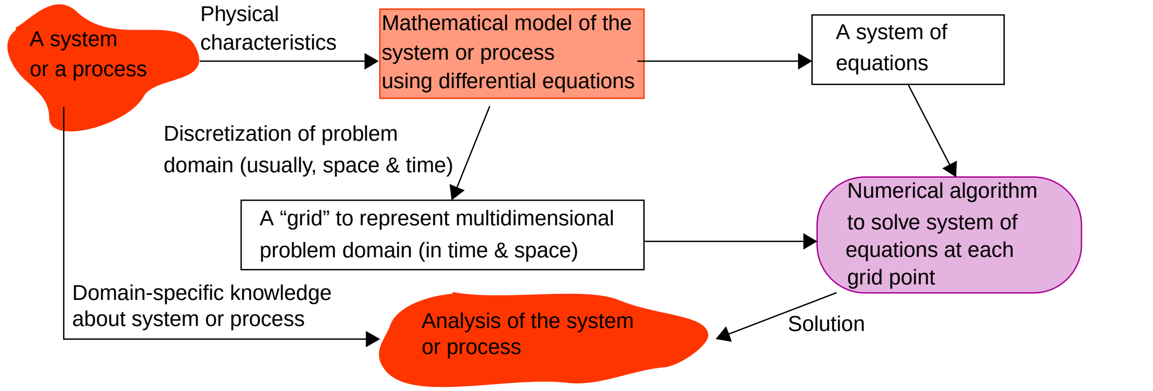

A system or a process
Physical characteristics
Mathematical model of the system or process using differential equations
A system of equations
Discretization of problem domain (usually, space & time)
A “grid” to represent multidimensional problem domain (in time & space)
Numerical algorithm to solve system of equations at each grid point
Domain-specific knowledge about system or process
Analysis of the system or process
Solution

Figure 1. Overview of numerical solution of a typical CFD problem on a computer.

Numerical PDE solvers for CFD applications exhibit some distinguishing characteristics that are common among them. Some of these computational characteristics are noted in the following:

1. **Temporal Recurrence**: Usually a small kernel (i.e., solution algorithm) dominates the computation. Execution of this code is repeated over temporal or spatial locations of the problem domain.
2. **Memory Bound**: Most numerical solution techniques used in CFD applications employ sparse matrices to represent system parameters obtained from the differential equations. Therefore, data structures usually store non-zero elements of the original matrices to reduce memory usage. This implies that the data structures are full but each element is accessed several times during main solution process.
3. **Compute Intensive**: Typically, CFD solvers require more CPU cycles than communication bandwidth. Compute intensive work is dominated by memory loads/stores and floating point operations.
4. **Structured Matrices**: Many CFD applications are based on algorithms that operate on structured matrices. This results in repetitive and uniform memory accesses.

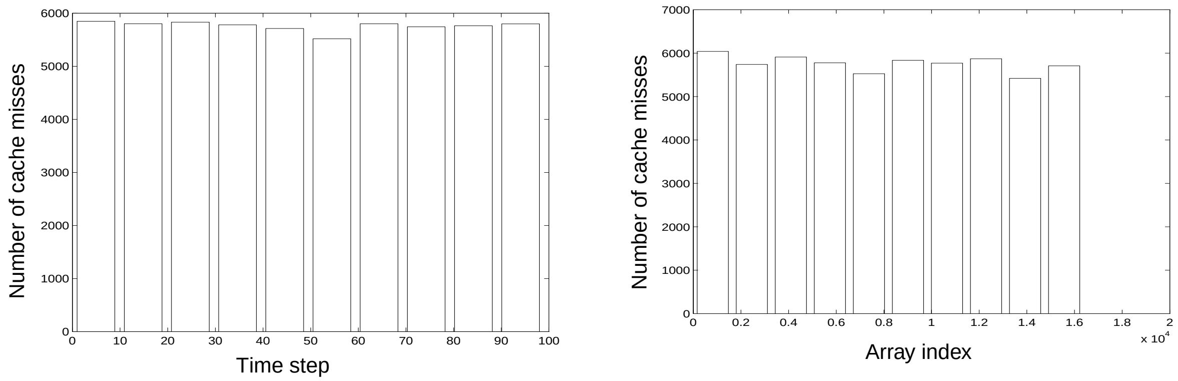
For cache-based HPC systems, the memory bound performance is the most important characteristic. In order to understand the effects of temporal recurrence and regular access patterns on cache performance, we executed a test code fragment (see Figure 2). We are interested in determining the number of cache misses over each time step as well as cache misses corresponding to accesses to each element of array A.

```
REAL A(1024, 1024)
INTEGER TEMP, TIME, I, J
REAL TEMP

DO TIME = 1:100
DO I = 1:128
DO J = 1:128
TEMP = A(I,J)
ENDDO
ENDDO
ENDDO
```

Figure 2. Example code.

We executed the example code of Figure 2 on one processor on an Origin2000. Each Origin2000 node consists of two processors, each with two levels of caches: a 64K primary cache (32K for data and 32K for instructions) and a 4MB secondary cache. In this example, we deliberately chose a matrix size that could not fit in the primary data cache. Additionally, the number of time steps was kept large, so that the “cold” cache misses in the beginning do not significantly skew the overall results. We used *ioctl* based interface to access R10000 performance counters and configured them to measure primary data cache misses [27]. Figure 3 shows the distribution of primary data cache misses over time steps as well as indices (calculated as I*128+J). This experiment suggests that recurrent and regular accesses to large arrays are expected to result in uniformly distributed cache misses over time and address space. Therefore, performance of typical CFD applications based on PDE solvers that run on cache-based systems is likely to be limited by the cache performance and utilization.



(a) **(b)**

Figure 3. Histograms of primary data cache misses.

For a measurement-based characterization of a class of CFD applications that solve PDEs, we selected following NAS parallel benchmarks: BT, SP, LU, FT, CG, and MG. These benchmarks represent different types of PDE solvers that are commonly used in real CFD applications. These benchmarks are executed on Origin2000, SP-2, and a cluster of PCs, which belong to three classes of HPC architectures: DSM system, distributed memory parallel systems, and distributed systems, respectively. Since explicit message-passing is the only common remote memory access approach among these systems, we use MPI-based implementation of NPBs for this workload study.

3 A Schema for Workload Characterization

In terms of resource usage, CFD workloads of interest to this study are typically compute-intensive PDE solvers. An application running on multiple processors needs inter-process communication, either through explicit message-passing or shared memory. Since this workload study focuses on explicit message-passing based applications, overall workload can be considered as alternating between computation and communication states, as shown in Figure 4. This represents the highest level of workload behavior that can be used to describe different types of scientific applications in a generic manner. There are three temporal components of this characterization that will be of interest for measurement-based studies in Section 4: (1) inter-arrival time between successive communication phases; (2) length of a computation phase; and (3) length of a communication phase. Since these quantities are non-deterministic, we will be interested in finding an appropriate probability distribution to specify them.

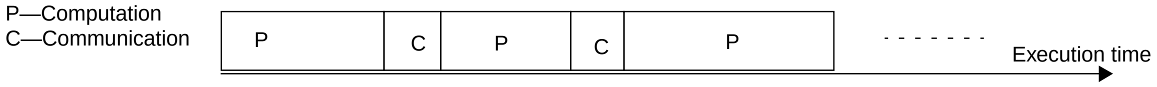


Figure 4. Alternating computation and communication phases in a typical CFD solver workload.

In order to extend the high-level workload characterization scheme of Figure 4, we consider further details of the computation phases. In case of a PDE solver for a CFD application, computation is dominated by floating point operations as well as memory accesses (primarily, memory loads). Figure 5 represents a more detailed model of a typical workload states running under a Unix-like operating system and four system resources: CPU, memory hierarchy, I/O, and network. The dashed lines help correlate this model with the overall characterization of Figure 4. CPU and memory access states correspond to a computation phase. Assuming no synchronization overhead, a communication phase requires two resources: CPU to make the system call for invoking a communication service and network resources for actual data transfer.

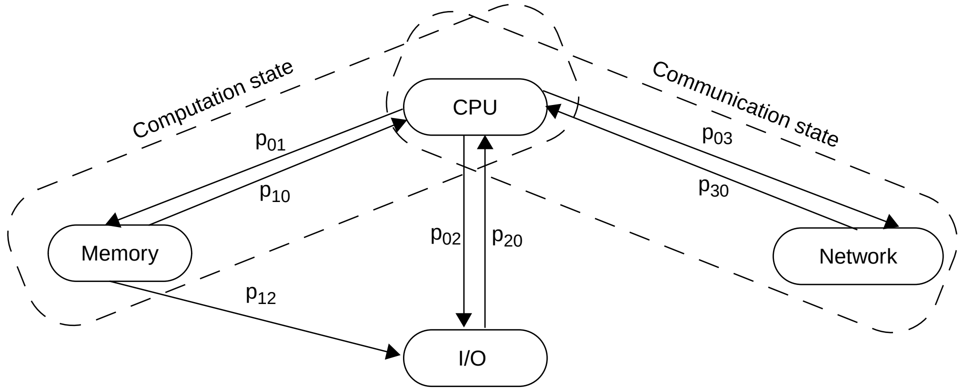


Figure 5. Possible states of a typical workload running under a Unix-like operating system.

CFD applications of interest to this study are based on PDE solvers, which typically do not involve intensive I/O operations. Therefore, we can eliminate the I/O state from our model and further expand on the memory access state to appropriately capture the resource usage behavior on a cache-based system. These changes are reflected in Figure 6. For shared memory multiprocessors, the network accesses correspond to the accesses to remote memory locations. Since we are focusing on explicit message-passing even of shared-memory systems, remote memory accesses are not considered as a part of load/store state. For programs that benefit from global address space on shared-memory systems, we could further eliminate a separate network access state and add it as another state under memory access.

Workload model can be parameterized for a specific system based on actual measurements. Parameterization involves: (1) determining a suitable probability density functions (*pdfs*) to represent the lengths of computation and communication states; (2) fitting a suitable *pdf* to represent inter-arrival time of successive communication states; (3) ranges of transition probabilities to each state within memory access state (e.g., primary cache miss rate); and (4) fraction of load/store instructions among all instructions executed.

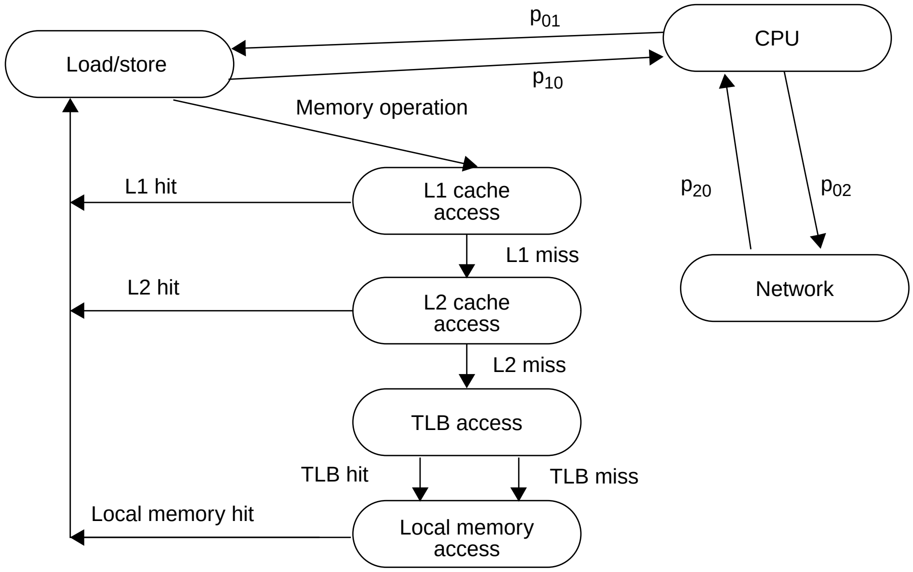


Figure 6. Detailed workload characterization scheme.

Using this scheme of workload characterization and parameterization on particular HPC systems, we can model and evaluate the performance of a parallel or distributed computing system under different operating conditions. Each nodes of the system may execute one or more processes that share the local resources: CPU, memory hierarchy, and network, according to the management policies enforced by operating system. Execution of an application process can be modeled by alternate requests for CPU and network time as illustrated by the process behavior given in Figure 7. The number of memory accesses and cost incurred due to cache misses is considered as a part of the computation phase time. Clearly, this level of detail does not capture many architectural details and may not be suitable for designing a new architectural feature in a microprocessor. However, this level of detail is sufficient for analyzing system level performance evaluation of a concurrent system. This level of detail is a compromise between accuracy of performance evaluation and amount of effort needed to use a detailed workload that captures instruction level behavior of a process and system resources.

In section 5, we follow the workload characterization schema presented in this section to design measurement-based experiments on three platforms of interest. Measurements yield quantitative ranges of workload parameters that are useful for modeling-based evaluation.

```
1. Set current_state = COMPUTATION
2. Set next_communication_time = interArrivalTimeDist.draw()
3. Set current_computation_length = next_communication_time - current_time
4. Determine the number of memory accesses using the total number of floating
   point operations for this program and determining the proportional number
   of floating point operations during current computation phase, assuming
   that all floating point operations are uniformly distributed during
   execution
5. Occupy the CPU for current_computation_length time and calculate the
   number of accesses to caches and memory according to their access and miss
   rates during current_computation_length
6. Set current_communication_length = communicationLengthDist.draw()
7. Set current_state = COMMUNICATION
8. Occupy the CPU for current_communication_length time
9. Asynchronously occupy the network resources for a fixed time
10. Repeat from 1 as long as current_time < simulation_end_time
```

Figure 7. A model for a PDE solver process behavior based on workload characterization.

4 Measurement-Based Experiments

Measurement-based workload characterization by gathering a similar set of performance statistics across heterogeneous platforms requires formidable effort due to differences in architecture, operating environments, and available monitoring tools. Three systems of interest to this study are based on following processors on each node: MIPS R10000, IBM RS/6000, and Intel Pentium Pro. These processors share two common architectural features that are important to note:

1. they support special on-chip counters to collect statistics related to processor operations; and
2. memory performance relies on the utilization of one or two levels of cache.

These features can be identified on many other state-of-the-art processors. Our workload study recognizes these common features by utilizing them for collecting performance measurements, in particular about the memory performance.

In order to provide a comparative measure of performance of three platforms, we consider the execution times of an MPI-based implementation of NPBs on these systems. Table 1 lists the execution times of each class A NPB on three platforms. These execution times reflect the wall-clock times excluding the initialization and final printing of result phases as coded in NAS implementation of these benchmarks [2]. Clearly, Origin2000 and SP-2 exhibit superior performance compared to the cluster of PCs due to their finally tuned and carefully designed parallel architectures.

Table 1. Comparison of class A NPBs' execution times on three platforms using four nodes.

Platform	Processor on each node	Processor speed (MHz)	Benchmark execution time (sec)					
			BT	SP	LU	FT	CG	MG
Origin2000	*R10000*	*195 MHz*	*820.34*	*392.92*	*364.90*	*45.87*	*13.21*	*16.90*
SP-2	*POWER2*	*66 MHz*	*593.61*	*472.72*	*478.73*	*66.73*	*10.77*	*16.30*
Cluster of PCs	*Pentium Pro*	*200 MHz*	*1,769*	*1,415*	*1,090*	*480.77*	*46.61*	*55.16*

In the following subsections, we briefly overview the architectures in addition to presenting workload studies carried out on these platforms. Results of a workload study on each platform is presented from three perspectives: overall statistics related to NPB executions; memory subsystem performance; and characterization of temporal resource usage behavior.

4.1 Origin2000

SGI Origin2000 is a Distributed Shared Memory (DSM) system with a cache coherent Non Uniform Memory Access (ccNUMA) architecture [20]. Each node of the system consists of two MIPS R10000 processors with two levels of separate data and instruction caches for each processor; and 4GB of main memory shared between two processors on a node. Multiple system nodes are connected in a hypercube topology through a high speed network. This system represents an important class of shared-memory platforms that are becoming popular in high performance computing because they offer ease of programming due to a global address space and scalability to large number of nodes.

Native SGI tools, such as Perfex and SpeedShop, benefit from the on-chip performance counters of R10000 processors for low-overhead and reliable measurements of processor operations [27]. We use these tools in addition to a custom tracing library to collect runtime information necessary for workload characterization. Statistics provided in the following subsections are collected during the execution of entire program, including initialization and final printing of results phases. These experiments differ from those whose results are reported in Table 1, which were based on the execution time of the benchmark portion of the code.

4.1.1 Overall Statistics

Table 2 provides the overall statistics of computation and communication phases obtained through tracing of class A NPBs. We inserted instrumentation at the boundaries of communication phases of each program to obtain the temporal characteristics from each of the four nodes where benchmark processes execute. On-chip cycle counter of R10000 processors were used to determine the timestamps. Results are based on the traces obtained from one of the four nodes; they are similar to the traces obtained from other nodes. Mean inter-arrival time between communication phases is small except in the case of FT benchmark that performs small number of global communication operations. Communication operations are scattered throughout the execution for other benchmarks. Origin2000 uses SGI's implantation of MPI message-passing library that benefits from the shared-memory architecture of this platform.

Table 2. Statistics of communication and computation phases of NPBs (class A) on Origin2000 obtained through custom instrumentation.

Benchmarks	Inter-arrival time of communication phases			Length of communication phases			Length of computation phases		
	Mean (sec)	Min (msec)	Max (sec)	Mean (sec)	Min (msec)	Max (sec)	Mean (sec)	Min (msec)	Max (sec)
BT	0.21	0.024	1.53	0.02	0.014	0.49	0.19	0.009	1.53
SP	0.05	0.029	0.80	0.004	0.014	0.12	0.04	0.014	0.79
LU	0.006	0.028	0.61	0.0004	0.018	0.17	0.005	0.008	0.59
FT	3.19	1.50	5.55	0.49	0.10	1.14	2.69	0.90	5.55
CG	0.009	0.37	0.70	0.001	0.02	0.04	0.008	0.12	0.70
MG	0.01	0.01	1.62	0.001	0.003	0.09	0.01	0.01	1.62

4.1.2 Memory Performance

Memory performance plays a key role in obtaining high performance for an application executed on a DSM system like Origin2000. There are four levels of memory hierarchy: primary cache, secondary cache, local memory shared between two R10000 processors on a node, and memory located on a non-local node. A data hit in primary cache takes only one clock cycle; a hit in secondary cache takes 10 cycles; a hit in the local memory consumes 100 cycles; and a hit in a non-local memory wastes well above 150 CPU cycles [20]. Thus the performance of a memory bound workload such as a PDE solver for a CFD application will suffer when program code does not fully utilize the caches.

Table 3 lists the ratios of memory access operations to all of the floating point operations. These statistics are obtained by executing each benchmark through Perfex on all four nodes. Perfex multiplexes the two on-chip counters of R10000 to measurements 31 different CPU and memory related metrics using a sampling-based approach. To account for software multiplexing of hardware counters and sampling, the resulting statistics are approximated through the counts. These approximations do not significantly affect the accuracy of measurements for sufficiently long-running programs. Based on these measurements, we can observe that almost every floating point instruction in these benchmarks require at least one memory access. This observation is consistent with common belief that CFD solvers are often memory bound applications [2]. These measurements also indicate that more than 80% of execution time is spent in accessing memory where execution time is monitored for the entire program. These memory accesses include floating point as well as integer instructions. These percentages are calculated by Perfex based on the total costs of cache and memory accesses and execution time. This information is important for performance tuning scenarios where we have to isolate the bottleneck resource. In case of a CFD solver application, the bottleneck resource is almost always memory. Number of floating point operations per second on each node represents a measure of computation-intensity of the workload on a given system. A larger value indicates that the application is computation intensive and likely to be memory- and CPU-bound.

Table 3. Memory performance statistics of NPBs on Origin2000 obtained by using Perfex tool.

Benchmarks	Execution time (sec)	MFLOPS per node	%age of memory access time to execution time	Ratio of memory access operations to floating point operations
BT	*794.22*	*33.87*	*0.83*	*1.33*
SP	*385.86*	*35.07*	*0.96*	*1.38*
LU	*353.63*	*59.96*	*0.91*	*0.93*
FT	*51.98*	*30.61*	*1.00*	*1.58*
CG	*13.81*	*15.91*	*1.00*	*3.58*
MG	*25.05*	*46.01*	*1.00*	*1.31*

Table 4 presents the cache and TLB miss statistics with the cost of these misses for each of the NPBs executed on Origin2000. The miss rates are expressed as a ratio of cache or TLB misses to the total number of memory references. Total number of memory references are very close to the number of floating point instructions for this workload, as suggested by the statistics of Table 3. Costs of cache or TLB misses are determined as percentages of time to replace cache line or TLB entries to the entire execution time of a program. For most of these cases, cache misses account for about 60% of the execution time. TLB misses are small due to the distribution of data among processors in a message-passing paradigm, which has a side-effect of reducing the effective size of data for each processor. Non-local memory accesses are through explicit message passing and its cost is considered as a part of communication cost according to workload characterization scheme presented in Section 3.

Table 4. Cache and TLB misses for NPBs on Origin2000 with 195 MHz R10000 processors obtained through Perfex.

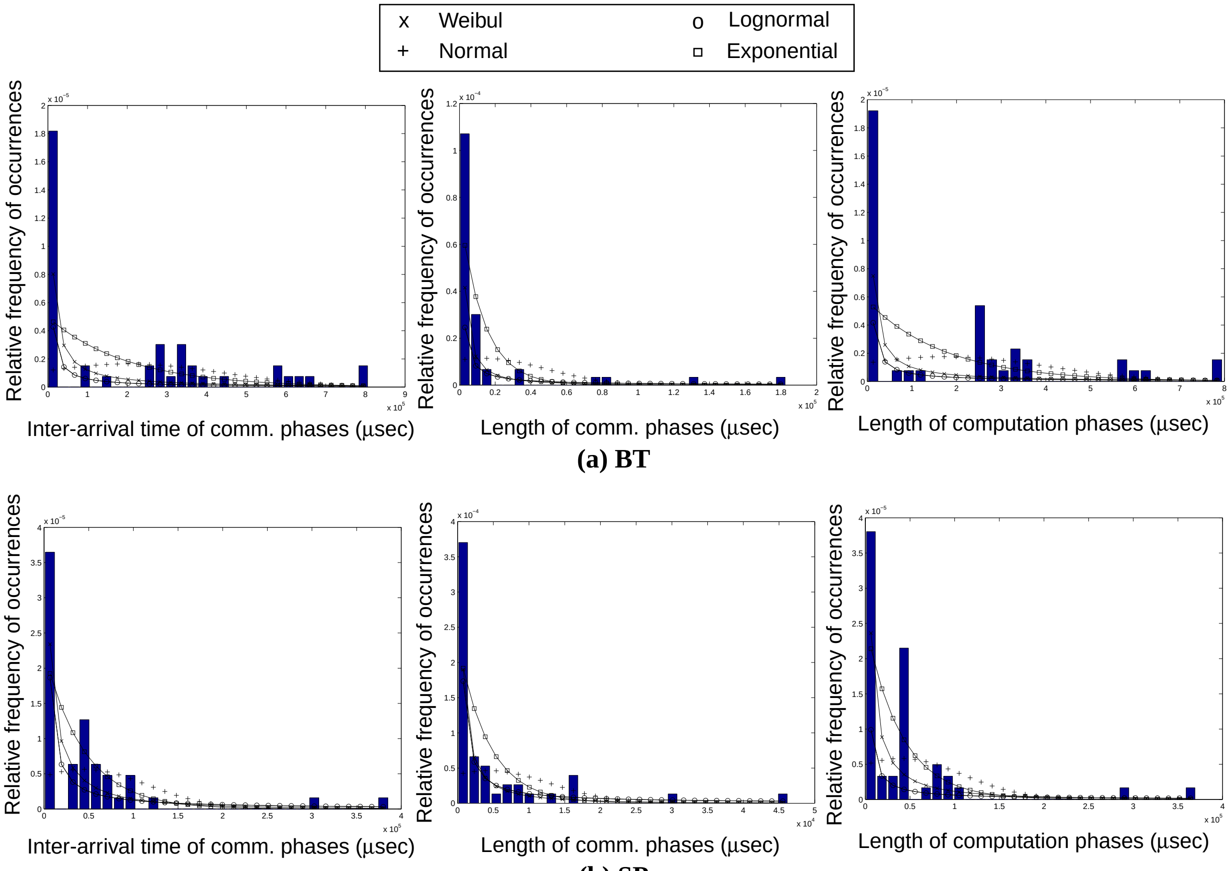
Benchmarks	**L1 data cache**		**L2 data cache**		**TLB**	
	Miss rate (%)	**Cost (%)**	**Miss rate (%)**	**Cost (%)**	**Miss rate (%)**	**Cost (%)**
BT	*11.9*	*21.81*	*21.50*	*39.28*	*0.10*	*9.09*
SP	*14.72*	*32.66*	*13.56*	*37.11*	*0.17*	*2.08*
LU	*14.03*	*36.10*	*6.39*	*19.34*	*0.35*	*7.35*
FT	*24.11*	*56.60*	*4.13*	*19.60*	*0.03*	*0.35*
CG	*26.63*	*69.03*	*5.78*	*33.45*	*0.13*	*0.75*
MG	*13.37*	*38.23*	*11.22*	*35.94*	*0.02*	*0.37*

4.1.3 Characterization of Temporal Process Behavior

Average resource usage statistics can only present a summary of process behavior and are not useful for modeling and evaluation scenarios that evolve over time. Probability distributions can adequately represent the temporal behavior of a process under study. A detailed simulation-based study can use appropriate probability distribution function (pdf) to model the requirements of computation resources over time from an application process. In this subsection we determine appropriate pdfs to represent alternating computation and communication phases for our reference PDE solver workload. For this purpose, we insert instrumentation in NPBs at the boundaries of each computation and communication phase. Analysis of each trace file yields a number of samples of three metrics: (1) inter-arrival time of communication phases, (2) length of communication phase, and (3) length of computation phase metrics.

Figure 8 presents histograms of samples of three metrics of interest collected during the execution of BT and SP benchmarks. These histograms are plotted such that the area under each histogram is equal to one. We can then compare the relative frequencies of measured values of the metrics to four theoretical pdfs: weibul, normal, lognormal, and exponential. We used statistics toolbox of Matlab to determine the values of these pdfs for each value of the three metrics. For each of the histogram, it can be observed that most of the occurrences are clustered around smaller metric values. This indicates that an exponential distribution is appropriate to represent these metrics. A closer examination of histograms of Figure 8(a) and (b) indicates a comparatively larger peak for the smallest bin, which is closely tracked by the theoretical values of lognormal pdf. However, an exponential distribution is an appropriate choice to represent inter-arrival times because it fits a range of small inter-arrival times in addition to approximately accounting for the initial peak. Additionally, computation and communication phases represent a fairly coarse-grained classification of the behavior of an application process. Therefore, use of exponential distribution ensures that the lengths of successive communication or computation phases are relatively independent due to the "forgetfulness" property of an exponentially distributed stochastic process[24]. Other research efforts have also used exponential distribution when workload is classified as alternating computation and communication processes [19].

Figure 9 represents the plots of normalized histograms and theoretical pdfs for four kernel benchmarks: LU, FT, CG, and MG. Except for FT, all other benchmarks exhibit a relatively large number of occurrences of small metric values, which is a behavior appropriately represented by an exponential distribution. In case of FT, there is a total of 17 communication phases during the entire benchmark execution. Therefore, the times for communication and computation are spread out to a wider range of values. In cases where a large number of samples are not available to fit a theoretical pdf, a normal distribution is considered a "robust" choice. However, we treat the temporal behavior of FT as a special case because a real CFD application typically runs for a long time and will have many more alternating computation and communication phases, which are likely to follow the exponential distribution as exhibited by all the other benchmarks in NAS suite.



(a) BT

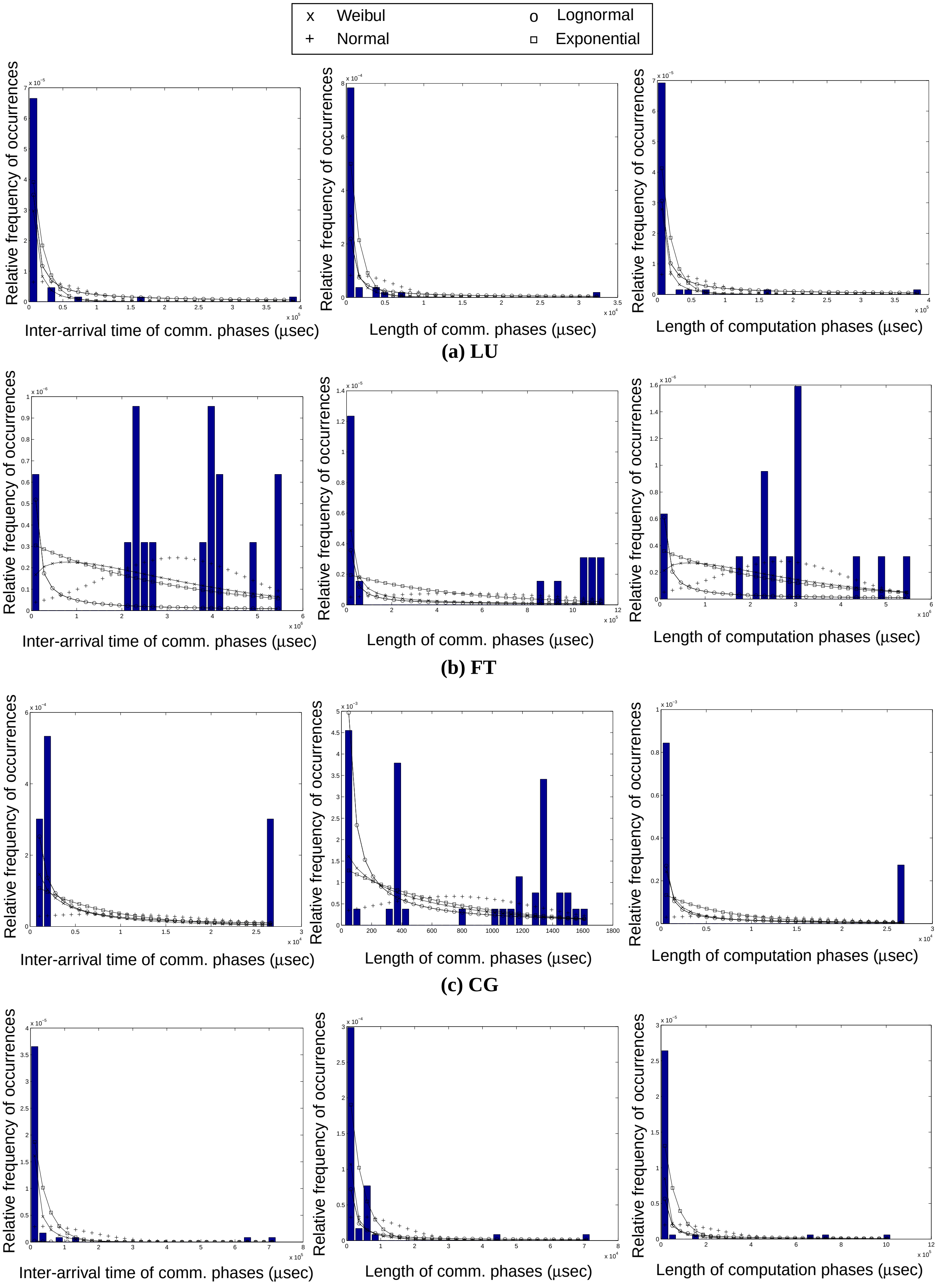
(b) SP

Figure 8. Histograms and pdfs for (a) BT and (b) SP benchmarks executed on four processors of an Origin2000.

From the measurement-based workload study on Origin2000, we conclude that:

1. PDE solvers exhibit alternating computation and communication phases over time, which can be represented by exponential distribution;
2. number of floating point operations per second per node range from 13 MFLOPS to 60 MFLOPS;
3. close to 80% of computation time is accounted for memory accesses;
4. primary level cache miss rate falls in the range of 11% to 27% with a cost of 10 CPU cycles for each miss;
5. secondary level cache miss rate lies in the range of 4% to 22% with a cost of 100 CPU cycles for each miss; and
6. TLB misses are insignificant for a message-passing program due to fine-grained data distribution.

These quantitative results can be used to parameterize a simulation model of an application process that executed on an Origin2000.



(a) LU

(b) FT

(c) CG

(d) MG

Figure 9. Histograms and pdfs for (a) LU, (b) FT, (c) CG, and (d) MG benchmarks executed on four processors of an Origin2000.

4.2 SP-2

An IBM SP-2 is composed of RS/6000-590 workstations (also known as, wide nodes) connected through a high speed switch to work as a distributed memory multicomputer system. Each RS/6000 node is based on a POWER2 multi-chip processor with 66.7 MHz clock, 256KBytes of data cache, two integer computation units, and two floating point computation units. Each cache line is 256 bytes wide. A key feature of RS/6000 wide node is that the main memory to cache bandwidth is same as POWER2 cache to processor bandwidth, which is equal to four 64-bits words per clock cycle. This feature allows a node to run close to peak (250 MFLOPS) performance rate. Nodes are connected by an omega network, which is a hierarchy of cross-bar switches. This switch can transfer data between SP-2 nodes at a latency of 1 microsecond and 40 MBytes/sec bidirectional bandwidth. Considering software overhead, the latency becomes 45 microseconds and bandwidth about 34 MBytes/second. Messages are transferred using buffered wormhole routing.

Like MIPS R10000, POWER2 architecture of an RS/6000 node also supports hardware counters to measure several processor activities, such as number of cache misses, TLB misses, and floating point instructions. We use an independently developed data collection tool for SP-2, called Hardware Performance Monitor (HPM) to gather the above statistics. Additionally, we use our instrumentation library to collect trace data from NPBs executed on SP-2 using a native implementation of MPI message-passing library. The HPM does not directly provide the number of total memory accesses during an execution. Therefore, we calculate this value in an indirect (an approximate) fashion. We shall assume that the number of memory accesses are at least equal to the number of floating point operations. This fact is substantiated by the statistics reported for an Origin2000 node in Table 3. In other words, we are assuming that the ratio of memory accesses to the number of floating point operations is an architecture independent characteristic of a program and holds for other processors of interest to this study for identical program inputs and code. We recognize the approximate nature of this assumption but we have to use it to determine cache and TLB miss rates and compare in the absence of total number of memory access count.

4.2.1 Overall Statistics

Table 5 provides the overall statistics of computation and communication phases obtained through tracing of NPBs. Unlike instrumentation for Origin2000 executions, timestamps are obtained through MPI timing function. This required a change in our custom instrumentation library for execution on SP-2 while the instrumented code is same that was used on Origin2000. Therefore, the number of traces for each benchmark are also the same. These timing statistics reflect the execution of entire programs including the initial setup and final printing of results phases in addition to the execution of code that is used as benchmark. A comparison with the same statistics obtained by executing NPBs on Origin2000 that were reported in Table 2 indicates the consistency of two results.

Table 5. Statistics of communication and computation phases of NPBs on SP-2 obtained by custom instrumentation.

Benchmarks	Inter-arrival time of communication phases			Length of communication phases			Length of computation phases		
	Mean (sec)	Min (msec)	Max (sec)	Mean (sec)	Min (msec)	Max (sec)	Mean (sec)	Min (msec)	Max (sec)
BT	*0.15*	*0.05*	*3.69*	*0.004*	*0.028*	*0.07*	*0.15*	*0.022*	*3.69*
SP	*0.06*	*0.12*	*1.15*	*0.003*	*0.03*	*0.07*	*0.06*	*0.04*	*1.08*
LU	*0.01*	*0.06*	*1.42*	*0.0003*	*0.04*	*0.09*	*0.01*	*0.02*	*1.36*
FT	*4.35*	*0.45*	*15.79*	*1.23*	*0.18*	*12.54*	*3.12*	*0.20*	*14.64*
CG	*0.01*	*0.42*	*1.97*	*0.002*	*0.09*	*0.004*	*0.01*	*0.17*	*1.97*
MG	*0.04*	*0.05*	*42.81*	*0.01*	*0.02*	*6.69*	*0.03*	*0.02*	*42.81*

4.2.2 Memory Performance

Memory subsystem of a node of distributed-memory parallel system is not as complicated as that of a node on a DSM system because cache coherence problem does not occur. An RS/6000 supports a simple and efficient local memory hierarchy and a fast network to improve remote memory accesses through explicit message-passing. The cost of a cache miss for a 66 MHz POWER2 is at least 8 cycles and that of a TLB miss is at least 28 cycles. The summarized results of HPM provide the overall execution time, total number of floating point operations, and number of cache and TLB misses.

Table 6 presents the cache and TLB miss statistics for the NPBs executed on SP-2. These statistics are obtained by starting HPM just before program execution and stopping it immediately after the execution finishes. At that point, a summary of CPU and memory related statistics are obtained from HPM. Costs for data cache and TLB misses is calculated directly from the summary statistics by using 8 cycles and 28 cycles costs for cache and TLB misses, respectively. However, cache and TLB miss rates are calculated based on the assumption that the number of memory accesses is equal to the total number of floating point operations.

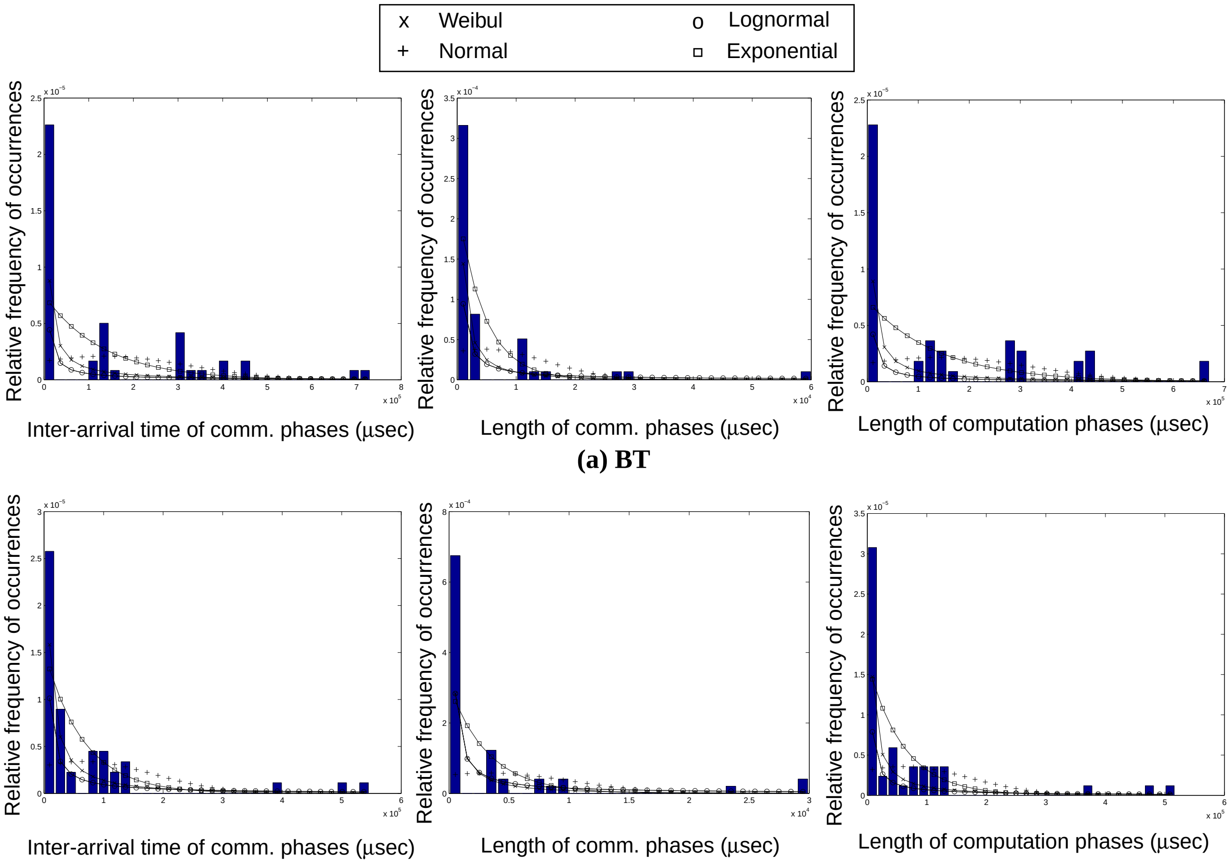
Table 6. Cache and TLB misses for NPBs on SP-2 with 66 MHz POWER2-based RS/6000 nodes obtained through HPM.

		Data cache		TLB	
Benchmarks	**MFLOPS per node**	**Miss rate (%)**	**Cost (%)**	**Miss rate (%)**	**Cost (%)**
BT	*64.75*	*1.48*	*11.60*	*0.10*	*2.62*
SP	*42.10*	*1.97*	*10.05*	*0.27*	*4.81*
LU	*59.88*	*0.95*	*6.92*	*0.0002*	*0.56*
FT	*39.08*	*1.05*	*4.99*	*0.08*	*1.27*
CG	*21.42*	*2.78*	*7.23*	*0.20*	*1.81*
MG	*7.02*	*2.97*	*2.52*	*0.12*	*0.35*

A comparison of cache miss costs for an SP-2 node with that of an Origin2000 node (see Table 4) indicates that an SP-2 node exhibits superior cache performance. The maximum cache miss cost is never higher than 12% of total execution time for an SP-2 node compared to a maximum of about 60% cost for an Origin2000 node. POWER2 and R10000 cache architecture have major differences. POWER2 supports 256KBytes cache with 256 bytes wide cache lines. An R10000 supports 32 KBytes primary data cache with 128 bytes wide cache lines and a secondary cache of 4 MBytes with 256 bytes wide cache lines. Apparently, the larger cache and block sizes of SP-2 nodes are favorable for the memory bound CFD solver workloads. Origin2000 nodes support smaller block size for primary cache reduce the false sharing problem due to globally shared address space. In addition, R10000 processor has a raw speed of 195 MHz, which makes the cost of a primary cache miss effectively lesser than the cost of cache miss of a POWER2 processor. Consequently, as shown in Table 1, the execution times of NPBs on two platforms do not show any consistent differences except for BT.

4.2.3 Characterization of Temporal Process Behavior

Figure 10 presents histograms of three metrics of interest for BT and SP benchmarks based on execution on four SP-2 nodes. Samples of these metrics are based on traces obtained from each node. There is no difference in the overall shape of these histograms compared to those obtained by execution on Origin2000 (see Figure 8). Largest number of inter-arrival times of communication phases, lengths of communication phases, and lengths of computation phases are small. Thus an exponential pdf can appropriately represent these temporal characteristics in this case also. The obvious difference is the parameters for the exponential pdfs because mean (and minimum and maximum) values for the three metrics differ from those obtained from Origin2000 execution.



(a) BT

(b) SP

Figure 10. Histograms and pdfs for (a) BT and (b) SP benchmarks executed on four nodes of SP-2.

Figure 11 represents the histograms and pdfs for four kernel benchmarks obtained by execution on four SP-2 nodes. Most of the occurrences of the metrics are clustered around small values except for FT due to very small number of communication phases. These temporal characteristics are again comparable with those obtained from Origin2000 execution with differences in parameters. These differences are due to the architecture as well as communication subsystems of two platforms.

As in case of Origin2000, we quantitatively summarize the results of workload study on SP-2 as following observations:

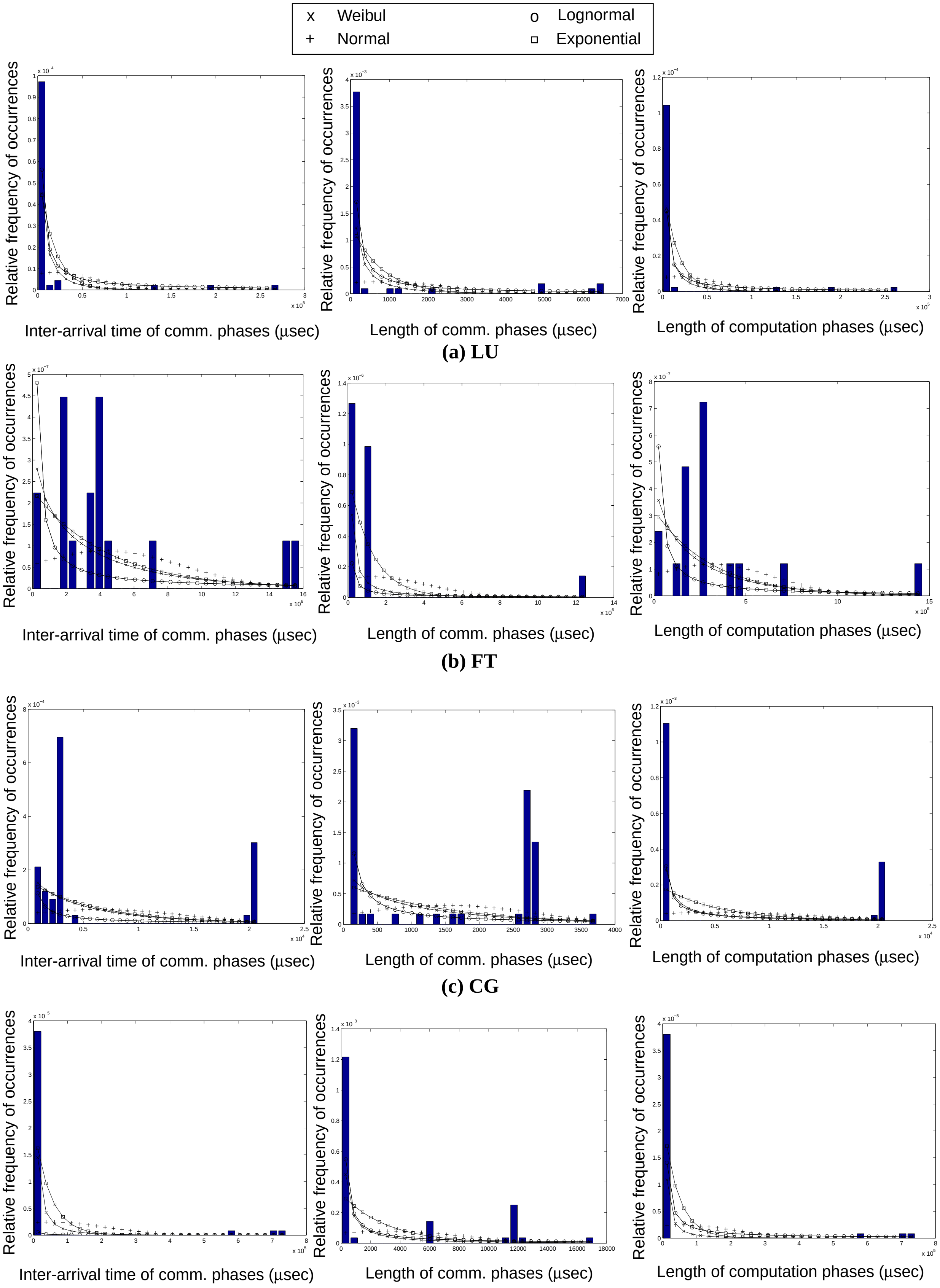
1. CFD applications based on PDE solvers exhibit alternating computation and communication phases over time, which can be represented by exponential distribution with appropriate parameters for each metric;
2. number of floating point operations per second per node range from 7 MFLOPS to 65 MFLOPS;
3. cache miss rate falls in the range of 1% to 3% with a cost of 8 CPU cycles for each miss;
4. TLB misses and their costs are again insignificant.

Cache performance of an SP-2 node is superior from that on Origin2000 as POWER2 processor considerably reduces the number of misses due to cache capacity. For a long-running CFD application, reduced number of cache misses can make a significant difference in its performance. This is one of the reasons of including memory performance as a part of our workload characterization effort.

4.3 Cluster of PCs

In recent year, the performance of personal computers (PCs) has caught up with the high-end graphical workstations. Due to economy-of-scale, PCs based on state-of-the-art CISC processors can have superior price-performance ratio compared to many workstations. This is one of the primary reasons for their popularity in a traditional Unix-based workstation market. As clusters of high-end workstation architectures were envisioned to provide performance comparable to supercomputers at fraction of price, clusters of high-end PCs can further reduce the cost of such a system.

In this subsection, we present measurement-based workload study on our in-house cluster of PCs using NPBs. This prototype system consists of 30 Intel Pentium Pro nodes running Linux and inter-connected through a Myrinet as well as a fast Ethernet networks. The raw speed of each processor is 200 MHz with two levels of cache. The first level is 8 KBytes and second level is 512 KBytes. Each node has 128 MBytes of main memory. Message passing is provided by MPICH implementation of MPI library. We use Portland



(a) LU

(b) FT

(c) CG

(d) MG

Figure 11. Histograms and pdfs for (a) LU, (b) FT, (c) CG, and (d) MG benchmarks executed on four nodes of SP-2.

Group's Fortran77 compiler for Linux to compile and link the NPBs.

4.3.1 Overall Statistics

Table 7 provides the overall statistics of computation and communication phases obtained through tracing of NPBs. The instrumentation library as well as the instrumented code is same that was used on SP-2. Therefore, the timestamps of events of interest are again obtained by MPI functions. These timing statistics reflect the execution of entire programs including the initial setup and final printing of results phases in addition to the execution of code that is used as benchmark. A comparison of the similar statistics obtained from Orgin2000 and SP-2 nodes indicates that lengths of same communication phases generally takes longer on PCs. Software overhead for communication is always a limiting factor for communication performance of a distributed computing system. The problem aggrevates when the system is developed based on unoptimized network interface layers of a non-commercial and unsupported operating system.

Table 7. Statistics of communication and computation phases of NPBs on a cluster of PCs.

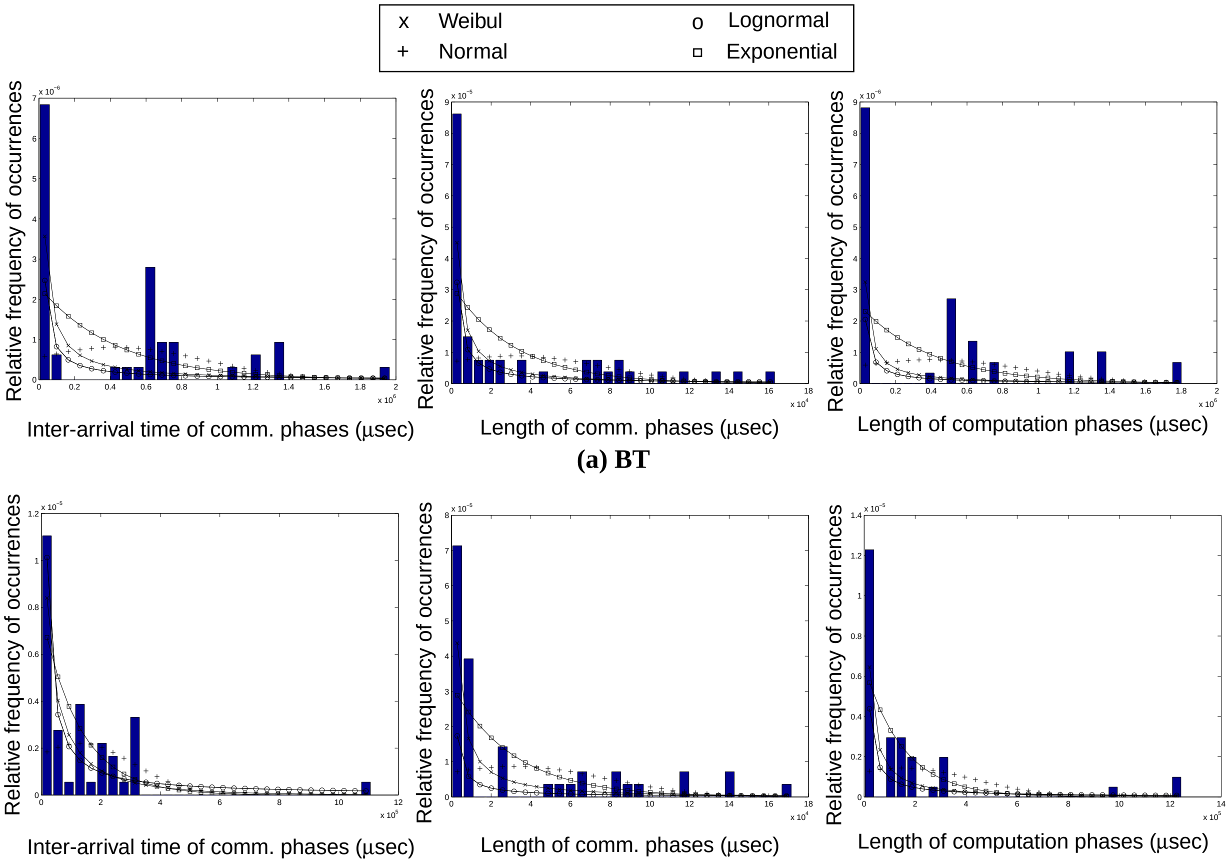
Benchmarks	Inter-arrival time of communication phases			Length of communication phases			Length of computation phases		
	Mean (sec)	Min (msec)	Max (sec)	Mean (sec)	Min (msec)	Max (sec)	Mean (sec)	Min (msec)	Max (sec)
BT	*0.44*	*0.08*	*4.96*	*0.03*	*0.03*	*0.17*	*0.42*	*0.05*	*4.94*
SP	*0.18*	*0.19*	*1.68*	*0.03*	*0.03*	*0.22*	*0.15*	*0.06*	*1.65*
LU	*0.02*	*0.21*	*1.38*	*0.001*	*0.04*	*0.25*	*0.02*	*0.08*	*1.28*
FT	*38.72*	*30.9*	*165.18*	*14.45*	*0.40*	*61.56*	*24.27*	*0.50*	*165.17*
CG	*0.04*	*0.59*	*16.06*	*0.01*	*0.17*	*0.08*	*0.03*	*0.33*	*16.06*
MG	*0.18*	*0.08*	*160.25*	*0.004*	*0.02*	*0.07*	*0.17*	*0.05*	*160.20*

4.3.2 Memory Performance

Pentium Pro also provides on-chip hardware counters to collect processor-level measurements. We use a publicly available tool, called *perfmon*, to obtain these measurements. At the time of writing this paper, we are installing a Linux patch to allow *perfmon* to monitor the CPU and memory statistics from each Pentimum Pro processor of selected nodes. We expect that the testing of this monitoring tool will take a few more weeks before we can obtain reliable resource usage information from this tool. We shall measure the MFLOPS rations, L1 and L2 cache misses and their costs for each NPB at that time.

4.3.3 Characterization of Process Behavior

Figure 12 presents histograms of three temporal metrics for BT and SP benchmarks executed on four nodes of PC cluster. These characteristics match approximately to those of Origin2000 and SP-2 executions. Largest number of occurrences are close to the small values of the metrics with a larger spread of values compared to Origin2000 and SP-2 observations. Additionally, lengths of communication phases are larger for these cases. Such communication characteristics are expected due to larger overheads of an operating system that is not particularly tuned for any specific parallel or distributed system.

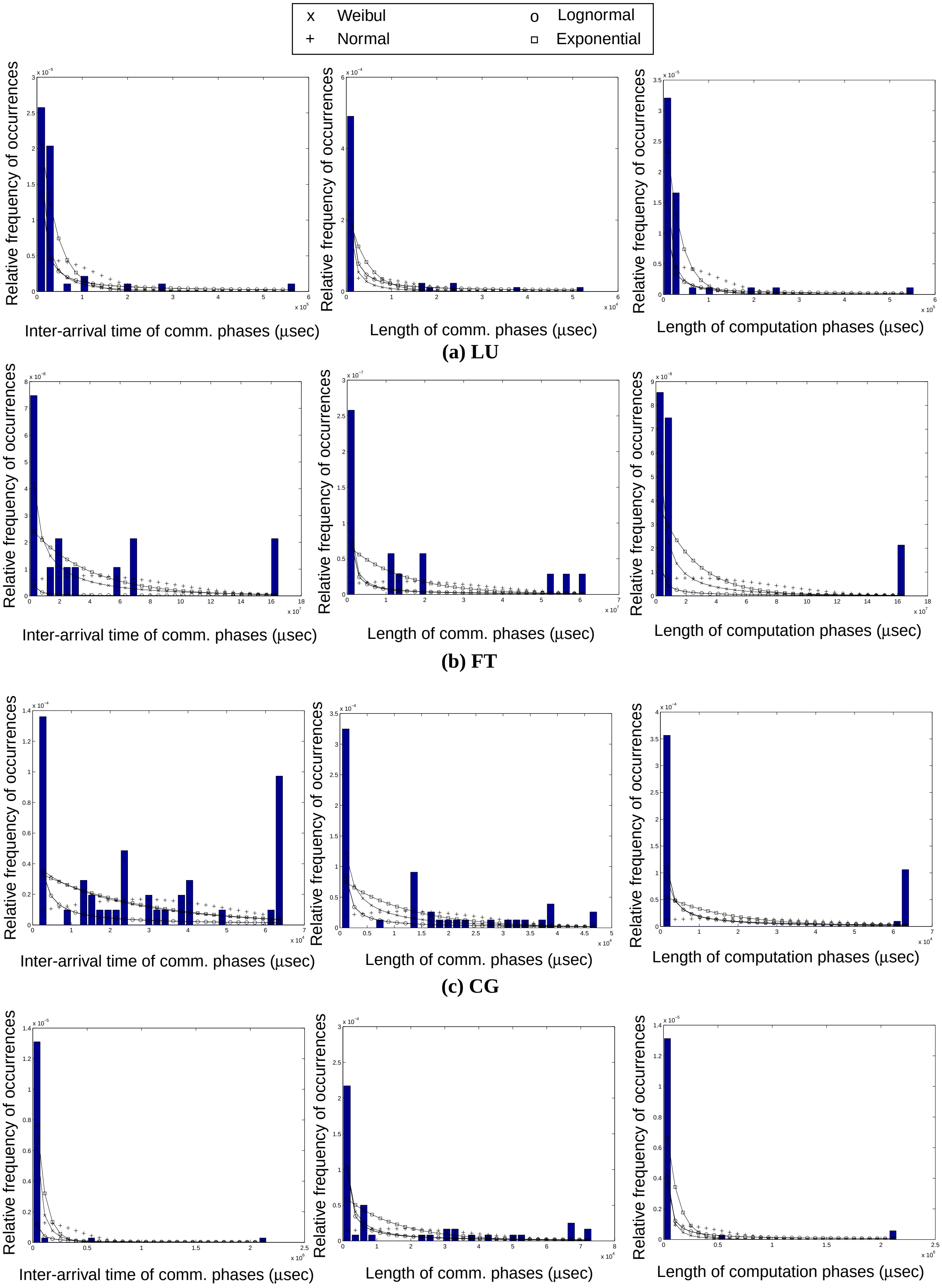


(a) BT

(b) SP

Figure 12. Histograms and pdfs for (a) BT and (b) SP benchmarks executed on four nodes of a cluster of Pentium Pro based PCs.

Figure 13 represents the histograms and pdfs of temporal metrics for four kernel benchmarks: LU, FT, CG, and MG. These results are consistent with the temporal characteristics of BT and SP. Histograms of three metrics closely follow the exponential distribution. Again, the lengths of time spent in communication phases are significantly larger compared to those on Origin2000 and SP-2 executions.



(a) LU

(b) FT

(c) CG

(d) MG

Figure 13. Histograms and pdfs for (a) LU, (b) FT, (c) CG, and (d) MG benchmarks executed on four nodes of a cluster of Pentium Pro PCs.

Temporal characteristics of NPBs are quantitatively summarize as following observations:

1. CFD solvers exhibit alternating computation and communication phases over time, which can be represented by exponential distribution with appropriate parameters for each metric; and
2. communication latencies are significantly higher due to untuned operating system software;

This concludes our study of CFD applications based on PDE solver workloads on three platforms. Depending on the system architectures and operating system software, the performance of memory and communication systems show significant differences. In order to use this characterization for performance modeling and evaluation, we need to account for these architecture- and operating system-based differences to accurately model a particular system. Coarse grain temporal characteristics of CFD solver workloads are identical across these platform with different parameters.

5 Putting Workload Characterization to Work

Workload characterization of PDE solver based CFD application is applicable to a number of performance evaluation scenarios as detailed in Table 8. Majority of these performance studies have one common thread: they emphasize on the ability to analyze performance at the entire system level rather than individual applications or system components level. We are primarily using CFD solver workload characterization to: (1) identify application performance tuning methodologies based on resource utilization profiles; and (2) evaluate the performance of heterogeneous metacomputing environments. We present a brief overview of these performance studies in the following subsections.

5.1 Performance Tuning

Measurement-based workload study of CFD solvers provided insight into the system resource behavior of these applications. One qualitative result of analysis presented in Section 4 is the memory-bound behavior of these applications, which are generally compute-intensive. If these applications result in efficient use of memory subsystem, performance can significantly improve.

In this subsection, we present our experience of performance tuning of sequential implementation of NAS benchmark BT on Origin2000. About 97% of the execution time during this execution is CPU time, which also includes memory access time. Additional measurements reveal that primary and secondary data cache

Table 8. Scenarios of applying CFD solver workload characterization.

Performance evaluation scenario	Explanation
Application performance tuning based on resource utilization profiles	*Performance of individual applications on a particular HPC platform are better understood when we know the characteristics of a larger class of similar applications on that system. Workload characterization was directly useful for understanding performance bottlenecks in general and memory performance bottlenecks in particular.*
Cache performance prediction	*We are in the process of designing a cache performance prediction tool that is based on generic workload characterization and system model. The main objective of this tool is to allow the user to study cache performance of basic blocks in the code.*
Performance studies of metacomputing systems	*We are using workload characterization to analyze the performance and throughput of metacomputing systems under different operating conditions.*
Effect of heterogeneous workloads on QoS	*Often high performance computing servers are used to execute a diverse mix of workloads. Workload characterization is useful to study QoS of such systems.*
Study of job scheduling policies	*Production high performance system for scientific workload are usually used in a batch mode. Workload characterization can be used quantitatively analyze various job scheduling policies.*
Design and evaluation of virtual shared memory systems	*Workload characterization can be used to fully analyze and tune virtual shared memory systems to deliver high performance when actually implemented.*
Design of data collection systems	*Monitoring systems can be designed to have low overhead to the system and application using workload characterizations that represent interactions among software and hardware resources of a system.*

misses are responsible for about 70% of the CPU time during this execution. In order to tune cache performance, we first identify the sections of code that account for largest proportion of execution time. Subsequently, we focus on this code and modify it to improve cache utilization.

Figure 14(a) shows a profile of primary data cache misses with respect to various subroutines of the BT benchmark. The graph represents the number of data cache misses within a subroutine as a percentage of total number of cache misses during the execution. Most of the data cache misses are clustered in a small number of subroutines. It implies that these parts of the code have relatively large number of memory accesses compared to others, and are more likely to incur overhead due to cache misses.

We optimized the memory accesses by modifying some loop nests in `compute_rhs` subroutine. Figure 14(b) shows the profile after tuning one subroutine that was responsible for largest number of primary data cache misses. The important thing to note in Figure 14(b) is that the relative memory access overhead of the most expensive subroutine has considerably reduced. This tuning results in about 15% improvement in overall execution time of BT.

This process of identifying a bottleneck resource and modifying the code to tune the usage of that resource

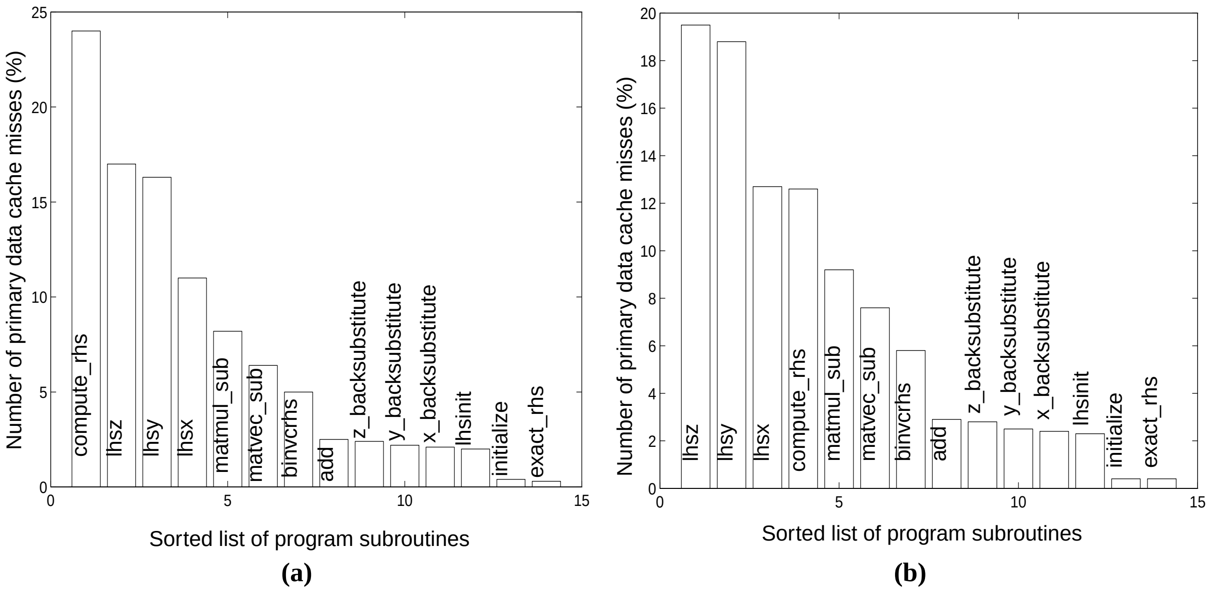


Figure 14. Profile of data cache misses for BT (a) before and (b) after cache performance tuning.

can be applied to other applications as well. In this scenario, workload characterization was helpful to provide us an *a priori* knowledge about possible bottleneck resource without resorting to aggressive measurements of actual application.

5.2 Modeling and Evaluation of a Metacomputing System

As illustrated by Figure 15, a metacomputing system typically consists of a set of high performance computing sites, data collection instruments, and large databases at geographically separate sites connected through a high bandwidth and low latency wide-area network. Other terms used for describing such a distributed system include *Information Power Grid*, *National Technology Grid* or *Computational Grid* due to similarities between a ubiquitous computing environment and an electric power grid. The goal of such systems is to make the computing resources widely available to users regardless of their geographical location. In addition, users should be able to focus on their application domains rather than dealing with the subtleties of porting and optimizing their codes to different high performance systems. See [8,11,18,23,25,26] for details about a number of projects addressing design and use of metacomputing systems.

Metacomputing systems are currently at planning and prototypical stages. Modeling and evaluation of

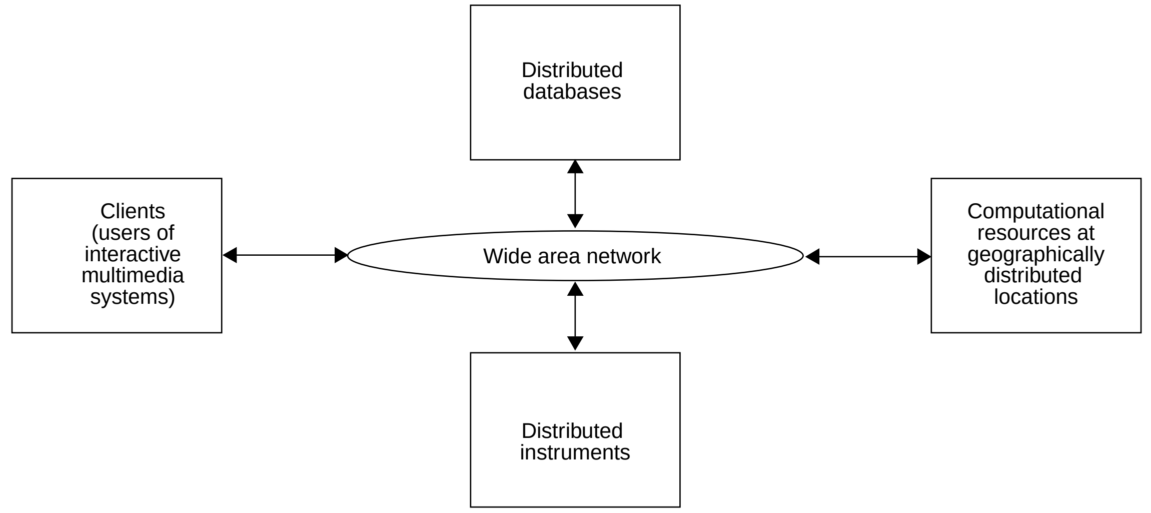

Distributed databases
Clients (users of interactive multimedia systems)
Wide area network
Computational resources at geographically distributed locations
Distributed instruments

Figure 15. A typical metacomputing system.

these systems at this stage is necessary to ensure that the production systems will be able to meet Quality of Service (QoS) requirements of specific applications. Although QoS is s generic problem, we focus on this problem in the context of resource management and scheduling in a heterogeneous environment. Czajkowski et al. identify five resource management issues that are specific to metacomputing systems: (1) site autonomy requirements; (2) heterogeneous local resource scheduling approaches and tools; (3) possibility of managing heterogeneous types of workloads; (4) co-allocation of resources for certain applications; and (5) adaptive control or resources to match dynamically varying application requirements and resource availability.

Our modeling effort focuses on two objectives: (1) quantitative comparison of different resource scheduling and management policies; and (2) impact of specific resource scheduling policies on domain-specific QoS metric. Figure 16 illustrates a queuing network model to represent a metacomputing system with resource management components. System is represented by four types of resources: computing and database server resources; distributed instruments; one or more resource managers; and a wide area network that connects all the resources. These resources are shared among multiple clients that occupy the resources interactively or in a batch mode. Components of this model labeled as HPC system models are parametrized using workload study presented in this paper to represent the execution of a CFD solver workload. For an interactive job, the QoS metrics are latencies for a transaction and number of missed deadlines. A smaller value of both of these metrics is desirable. For a batch job, the QoS metric is waiting time in a batch queue and measured floating point operations per second. A smaller waiting time indicates that the choice of an HPC system by resource manager to schedule the job is based on its current load. On the other hand, selection of a system based on its load may not result in high performance measured as a ratio of useful floating point operations during the execution time. A high value of this ratio is desirable. An optimal choice of a computing resource could be a compromise between waiting time and number of floating point operations achieved on that system.

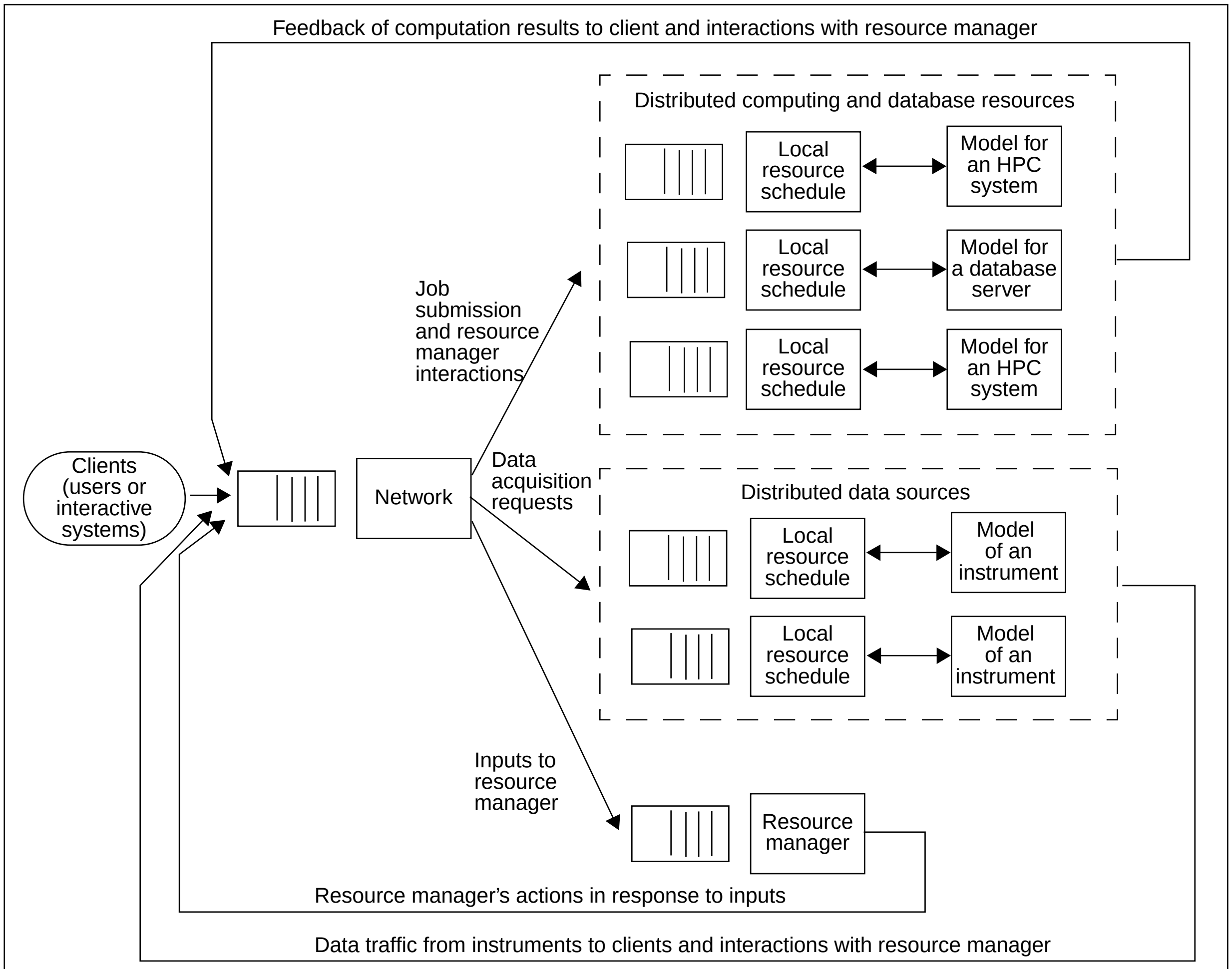

Feedback of computation results to client and interactions with resource manager
Distributed computing and database resources
Local resource schedule
Model for an HPC system
Local resource schedule
Model for a database server
Local resource schedule
Model for an HPC system
Job submission and resource manager interactions
Clients (users or interactive systems)
Network
Data acquisition requests
Distributed data sources
Local resource schedule
Model of an instrument
Local resource schedule
Model of an instrument
Inputs to resource manager
Resource manager
Resource manager's actions in response to inputs
Data traffic from instruments to clients and interactions with resource manager

Figure 16. A queuing network model for a metacomputing system with resource management.

Our modeling based evaluation efforts of metacomputing resource management systems continues. We are using a C++SIM based simulator to model different components of the model and then hierarchically integrate them for QoS based performance study under different operating conditions [3]. We are conducting a workload study of the local resource scheduling components, called *Portable Batch System* (PBS [13]) in terms of job waiting time as function of number of nodes requirement, execution time requirement, and time of the day. We shall use these system load characteristics to improve resource scheduling policies. We also intend to model a broad spectrum of resource management policies found in existing tools and testbeds including, Condor [22], CODINE [10], GUSTO [6], Legion [11], and Load Leveler [16] in addition to the PBS. Detailed quantitative evaluation of QoS metrics based on the models for these resource managements systems is useful for the software environment developers for the future metacomputing systems.

6 Conclusions

In this paper, we presented an outline of our CFD solver workload characterization effort and the scenarios where it is being applied. We also presented detailed workload characterizations from 6 NPBs on 3 HPC systems of interest. We outlined the use of this characterization to evaluate various “what-if” scenarios regarding the design of a metacomputing system. Based on measurements from an example, we also illustrate the use of workload characterization to identify performance bottlenecks in a program.

Workload studies indicate that the temporal characteristics are similar across different platforms. Differences in architecture and operating systems were also elaborated by the measurements-based studies. Origin2000 requies a high cache utilization for delivering close to the peak performance, which is not the case for an SP-2. Communication overhead is significantly higher for message-passing on a Linux PC compared to Origin2000 and SP-2. This workload study quantitatively determined these differences for NPBs.

Use of workload characterization to design and evaluate HPC systems has been limited due to the difficulty in collecting runtime information with minimum intrusion to the programs. However, with the availability of on-chip measurement support on state-of-the-art processors, it is possible to collect system-level information with minimum overhead [1]. Our workload characterization effort is an example where these measurements enabled various performance studies.

References

[1] G. Ammons, T. Ball, and J. Larus, “Exploiting Hardware Performance Counters with Flow and Context Sensitive Profiling,” *Proc. of ACM SIGPLAN Conf. on Programming Language Design and Implementation*, Las Vegas, Nevada, June 15-18, 1997.

[2] D. Bailey et al., “The NAS Parallel Benchmark 2.0,” Technical Report NAS-95-020, Dec. 1995.

[3] *C++SIM User’s Guide*, Draft Version 1.0, Dept. of Computing Science, University of Newcastle Upon Tyne.

[4] L. Cardelli, “Global Computation,” *ACM Sigplan Notices*, 1997.

[5] C. Connelly and C. Ellis, “Workload Characterization and Locality Management for Coarse-Grain Multiprocessors,” Tech. Report CS-1994-30, Dept. of Comp. Science, Duke University, Oct. 1993.

[6] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, “A Resource Management Architecture for Metacomputing Systems,” Technical Report, available on-line from: http://www.globus.org/globus/papers.htm.

[7] Robert T. Dimpsey and Ravishankar K. Iyer, “A Measurement-Based Model to Predict the Performance Impact of System Modifications: A Case Study,” IEEE Transactions on Parallel and Distributed Systems, 6(1), January 1995, pp. 28–40.

[8] Ian Foster and Carl Kesselman, “A Metacomputing Infrastructure Toolkit,” *International Journal of Supercomputer Applications*, 1997.

[9] J. Gee, M. Hill, D. Pnevmatikatos, and A. Smith, “Cache Performance of SPEC92 Benchmark Suite,” *IEEE Micro*, August 1993.

[10] GENIAS Software GmbH., “CODINE: Computing in Distributed Networked Environments,” 1995. Available on-line from http://www.genias.de/genias/english/codine.html.

[11] A. Grimshaw et al., “The Legion Vision of a Worldwide Virtual Computer,” *Communications of the ACM*, 40(1), Jan. 1997.

[12] G. Haring and G. Kotsis, “Workload Modeling for Parallel Processing Systems,” *Proc. of the 3rd Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS’95)*, Durham, North Carolina, Jan. 1995, pp. 8–12.

[13] R. Henderson and D. Tweten, “Portable Batch System: External Reference Specifications,” Technical Report, NASA Ames Research Center, 1996.

[14] John L. Hennessy, “Perspectives on the Architecture of Scalable Multiprocessors: Recent Development and Prospects for the Future,” *State of the Field Talk, High Performance Networking and Computing Conference (SC ‘97)*, San Jose, California, Nov. 15-21, 1997.

[15] Herman D. Hughes, “Generating a Drive Workload from Clustered Data,” *Computer Performance*, 5(1), March 1984.

[16] *IBM Load Leveler: User’s Guide*, 1993.

[17] K. Kennedy, “Programming Support Software for High Performance Computers,” *State of the Field Talk, SC ‘97*, San Jose, California, Nov. 15-21, 1997.

[18] Ken kennedy, Charles F. Bender, John W.D. Connolly, John L. Hennessy, Mary K. Vernon, and Larry Smarr, “A Nationwide Parallel Computing Environment,” *Communications of the ACM*, 40(11), November 1997, pp. 63–72.

[19] Leonard Kleinrock and Willard Korfhage, “Collecting Unused Processing Capacity: An Analysis of Transient Distributed Systems,” *IEEE Transactions on Parallel and Distributed Systems*, 4(4), May 1993, pp. 535–546.

[20] James Laudon and Daniel Lenoski, “The SGI Origin: A ccNUMA Highly Scalable Server,” *Proc. of the 24th Annual International Symposium on Computer Architecture*, Denver, Colorado, June 2-4, 1997, pp. 241–251.

[21] J. Larus, “The SPIM Simulator for the MPIS R2000/R3000,” in *Computer Organization and Design—The Hardware/Software Interface* by David A. Patterson and John L. Hennessy, Morgan Kaufmann Publishers, 1994.

[22] M. Litzkow, M. Livny, and M. Mutka, “Condor—A Hunter of Idle Workstations,” *Proc. of 8th International Conference on Distributed Computing Systems*, pp. 104–111.

[23] Daniel A. Reed, Roscoe C. Giles, and Charles E. Catlett, “Distributed Data and Immersive Collaboration,” *Communications of the ACM*, 40(11), November 1997, pp. 39–48.

[24] Sidney I. Resnick, *Adventures in Stochastic Processes*, Birkhauser, 1992.

[25] Larry Smarr, “Toward the 21st Century,” *Communications of the ACM*, 40(11), November 1997, pp. 29–32.

[26] Rick Stevens, Paul Woodward, Tom DeFanti, and Charlie Catlett, “From the I-WAY to the National Technology Grid,” *Communications of the ACM*, 40(11), November 1997, pp. 51–60.

[27] M. Zagha, B. Larson, Steve Turner, Marty Itzkowitz, “Performance Analysis Using the Mips R10000 Performance Counters,” *Proceedings of Supercomputing ‘96*, Pittsburgh, Pennsylvania, Nov. 1996.

NAS TECHNICAL REPORT

Title:

Workload Characterization of CFD Applications Using Partial Differential Equation Solvers

Author(s):

Abdul Waheed and Jerry Yan

Reviewers:

"I have carefully and thoroughly reviewed this technical report. I have worked with the author(s) to ensure clarity of presentation and technical accuracy. I take personal responsibility for the quality of this document."

Two reviewers must sign.

Signed: ______________________________

Name: _H. Jin____________________

Signed: ______________________________

Name: _M. Hribar________________

After approval, assign NAS Report number.

Branch Chief:

Approved: _____________________________

Date:

NAS ReportNumber:

NAS-98-011